Multi-Line Batch Scheduling by Similarity

Ignacio Arnaldo
MIT, CSAIL
Cambridge, MA
iarnaldo@csail.mit.edu

Erik Hemberg
MIT, CSAIL
Cambridge, MA
hembergerik@csail.mit.edu

Una-May O'Reilly
MIT, CSAIL
Cambridge, MA
unamay@csail.mit.edu

ABSTRACT

We introduce a real-world job shop scheduling problem where the objective is to minimize configuration costs that depend on the sliding pairwise similarity between two assets ordered one after the other in a processing batch. This implies that our fundamental challenge is to learn from the costs what constitutes asset similarity in the context of batching locally and optimizing a multi-line end to end. We present a 3 component scheduling system: simulator, scheduler and hyper-optimizer. The scheduler relies upon a machine learning algorithm – hierarchical clustering, to select, from an entry yard, assets for a batch based on weighted similarity. It then utilizes a weighted distance matrix to sequence the assets. The weights used by the scheduler are optimized online with an evolutionary algorithm.

Categories and Subject Descriptors

I.2.2 [**Artificial intelligence**]: Scheduling

Keywords

hyper-optimization; machine learning

1. INTRODUCTION

Late stage steel processing occurs on a very large scale: gigantic, heavy steel assets (coils or slabs typically) are chemically treated, rolled flat and/or galvanized by passing through what is termed a *multi-line*. A multi-line is a series of *components*, each consisting of an *entry yard*, *process* and *exit yard*. The exit yard of one process functions as the entry yard to the next. Groups of assets arrive from different sources at entry yards along the multi-line and can exit after a component. Scheduling is done in batches, locally at each component. Each asset transits through the multi-line independently, becoming a member of a different batch at each component.

Efficiently selecting the assets for a batch from the inventory yard and sequencing them is the challenge of our scheduling problem, see Figure 1. This problem is within the general class of job shop scheduling however it exhibits distinctions that make it atypical. For the multi-line to be cost-efficient (and other similar manufacturing multi-lines such as vehicle painting), the sequence of assets must be *smooth*. That is, the difference between any two assets, when one follows the other, must be minimized so that the process does not have to be adjusted (e.g., by heat change, chemical remixing or track movement). Rather than optimizing makespan or throughput, in this problem the primary objective is to reduce the cost of reconfiguring each process for the differing physical properties of each successive asset in a batch.

Similarity batching as selection and sequencing, henceforth simply called *batching*, is challenging for a number of reasons. Primarily, each asset is characterized by multiple physical properties. For example, a slab has three dimensions: length, width and thickness and is of a certain steel grade. The four properties are relevant to smooth sequences, one asset to the next, through the pickling process. The first three properties are relevant to smooth sequences through the tandem process but steel strength is an additional property that matters while grade does not. Figure 2 shows additional examples.

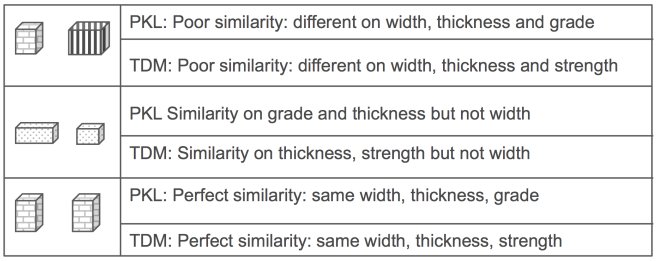

Figure 2: Examples of asset-to-asset dissimilarity

When a multi-line engineer chooses a batch of assets and sequences it, s/he must trade off differences in properties according to the influence each has on the cost of processing the sequence. Costs differ by properties, e.g., for pickling a difference in thickness is less or greater than the cost of a difference in width depending on the magnitude of the difference. At greater detail, depending on the size of an asset-to-asset property difference, the cost can be indexed by a level of severity defined as one of three levels: high, medium and low.

Because property differences and costs are computed on a sliding asset-to-asset basis, the sequencing problem is non-

GECCO'16 Companion, July 20-24, 2016, Denver, CO, USA

 ISBN 978-1-4503-4323-7/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2908961.2931647

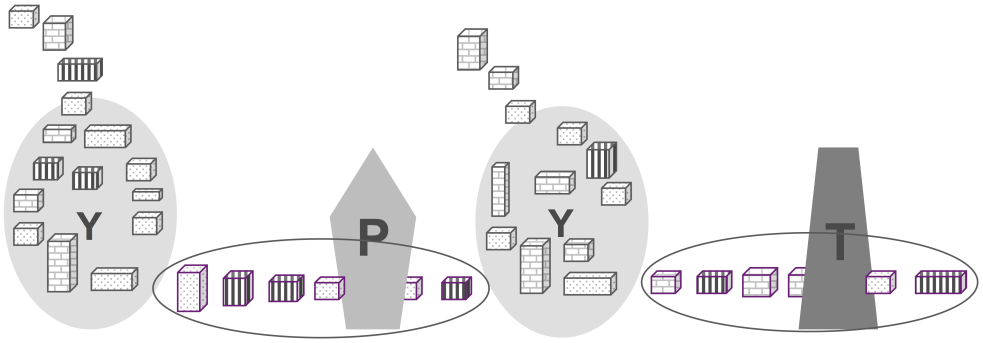

Figure 1: Multi-Line Scheduling. Yards = Y, Processes = P (Pickling) and T (Tandem). Asset properties are depicted visually. Batches are circled for illustration.

linear. Additionally, a *local* asset-to-asset pairing has a larger scale impact because it influences which assets are available for future batching. At yet a higher scale, while batch selection and sequencing is able to proceed independently at each component, components are linked because the exit yard of an upstream component can serve as the entry yard for a downstream one. Note that in our problem definition cost objectives do not directly express any costs spanning the entire series of the multi-line.

As a final complexity, the non-linearity sequencing dependencies for a process are different for each one given the nature of their work. Effectively, the complex concept of what makes two assets similar in one processing context is NOT the same as in another. Fundamentally, our challenge is to learn from process costs, *what constitutes asset similarity in the context of batching locally while considering the multi-line end to end.*

The overall contribution of this paper is the design and evaluation of a system solution to multi-line batching based upon similarity optimization, centered on a heuristic scheduler, see Figure 3.

There are a number of novel aspects to our solution:

1. The scheduler selects assets from the entry yard for a batch using *a machine learning clustering algorithm.* This integrates all property dimensions relevant to similarity. On our example multi-line definition and evaluation data, this is demonstrably superior to randomly selecting assets.

2. After it selects the batch, the scheduler sequences assets with a pairwise sliding window by referencing a weighted inter-asset distance score for each possible pair. These weighted inter-asset distance scores are returned by the clustering algorithm. We contribute two sequencing heuristics that use them: Nearest Neighbor (NN) and ranking by weighted distance (RANK). We compare them to simply sorting the assets on the most influential property (SORT) with our example multi-line definition and evaluation data. We find the Nearest Neighbor heuristic is significantly superior to the RANK heuristic and to sorting.

3. The weights that are integrated into the inter-asset distance score are intended to implicitly express asset similarity in terms of process costs. We contribute a means of optimally determining these weights. We make them parameters of the scheduler, terming them more precisely "hyper-parameters". To hyper-optimize them we use an evolutionary algorithm, CMAES [5]. The scheduler is executed multiple times (and each time it schedules numerous batches), each time with a different set of weights proposed by CMAES until the best set of weights (that produces the lowest cost schedules) is determined. This is shown in Figure 3

4. Conventionally, hyper-parameters are tuned once, with historical examples, then used indefinitely. We can refer to this as an *offline* paradigm. We offer a *online* paradigm which is feasible because we exploit cloud computing and parallelization. To tune online in our approach it is necessary to complete a run of CMAES that determines the weights for the next batch in the duration it takes to physically process the current batch. We simultaneously run CMAES 30 times on the cloud to determine the best weights and we also optimize the execution time of a run by multi-threading the framework's simulator. This allows us not only to tune the weights daily but also per batch, given our some simplistic assumptions about batch weight and corresponding asset quantity and processing time.

We proceed by first formally defining our problem in Section 2. We then we present related work in Section 3 before describing our system architecture in Section 4 and evaluating the scheduler in Section 5. Section 6 concludes with a summary. Finally, future work in Section 7.

2. PROBLEM DEFINITION

All symbols are defined in Table 1 for reference. For brevity, where unambiguous, we omit higher order superscripts. We are interested in minimizing the configuration costs associated with multiple components Υ_i each composed of an entry yard, process π, and exit yard:

$$\Upsilon_i = (Y_i^E, \pi_i, Y_i^X), \tag{1}$$

Components are linked serially into a multi-line Π

$$\Pi = (\Upsilon_1, \Upsilon_2, \Upsilon_3, \ldots \Upsilon_{|\Pi|})\ Y^E_{\Upsilon_i} = Y^X_{\Upsilon_j}, j > i \tag{2}$$

The costs of each process π depend on asset properties Φ specific to process π.

$$\Phi^{\pi^i} = (\phi_1^i, \phi_2^i, \ldots \phi_{|\Phi|}^i) \tag{3}$$

Cost severity, associated with a process and property $\mu^{\pi,\Phi}$ has 3 levels: hard H, soft S and penalty P. These levels are equivalent to high, medium and low.

$$\mu = (H, S, P) \tag{4}$$

Severity may simultaneously occur on multiple levels, e.g a major reconfiguration and minor reconfiguration may be required.

Because the configuration of a process must be adjusted before different assets pass through it, e.g. changing furnace temperature or dip chemical treatments, process costs are assigned on an asset-to-asset, a_0-to-a_1, basis when the values of a property are not similar. Described fully, a process cost, f is a function of process, property, severity, and the two assets. f maps to a real value.

$$y = f(\pi_i, \phi_j^i, a_1, a_0, \mu_k^{i,j}), y \in \mathbb{R} \tag{5}$$

A batch β is an ordering of assets selected from a yard. We presume that the quantity of assets in a batch is smaller than the inventory quantity.

$$\begin{aligned} \beta &= \{a^1, a^2, \dots a^{|\beta|}\} \\ |\beta| &<< |Y^E| \end{aligned} \tag{6}$$

We aggregate the asset-to-asset cost f for each property, severity and process over a batch β of assets a using a pairwise, overlapping, sliding window and then over successive batches B that run through the multi-line over a time duration.

$$cost(B, \pi, \phi, \mu) = \sum_{\beta}^{B} \sum_{a}^{\beta} f(\pi_i, \phi_j^i, a_1, a_0, \mu_k^{i,j}) \tag{7}$$

Our goal is to minimize the costs of manufacturing multiple sequences B by controlling the pairwise sequencing of assets. In other words to find the B that minimizes all the costs within (7). For practicality, to reduce the objective space we will weight the cost of a property by a coefficient of severity, K_H, K_S and K_P.

$$\begin{aligned} f(\pi, \phi, a_1, a_0) =& K_H f(\pi_i, \phi_j^i, a_1, a_0, \mu_H^{i,j}) + \\ & K_S f(\pi_i, \phi_j^i, a_1, a_0, \mu_S^{i,j}) + \\ & K_P f(\pi_i, \phi_j^i, a_1, a_0, \mu_P^{i,j}) \\ K_H >> K_S >> K_P & \end{aligned} \tag{8}$$

Then we sum the costs of a property ϕ_ζ across processes.

$$\begin{aligned} cost(\phi) = f(\pi_i, \phi_\zeta^i, a_1, a_0) + f(\pi_j, \phi_\zeta^j, a_1, a_0) \\ \pi_i \neq \pi_j \end{aligned} \tag{9}$$

To explain the notation, it expresses that π_i and π_j are different processes while ϕ_ζ is the same property. An example is helpful here. Suppose the multi-line has 2 processes, PKL and TDM, and the width difference between each pair of assets is independently relevant to each of them. There are hard, soft and penalty severities in cost. We sum the width difference costs weighted by severity of each of PKL and TDM before adding them together.

Table 1: Symbols in problem definition

Symbol	Meaning
$\Upsilon_i = (Y_i^E, \pi_i, Y_i^X)$	component with entry yard, process and exit yard
$\Pi = (\Upsilon_1, \Upsilon_2, \Upsilon_3, \dots \Upsilon_{\lvert\Pi\rvert})$	multi-line
$Y_{\Upsilon_i}^E = Y_{\Upsilon_j}^X, j > i$	component coupling
$\beta = \{a^1, a^2, a^3, \dots, a^{\lvert\beta\rvert}\}$	batch of assets
$B = \{\beta_1, \beta_2, \beta_3, \dots, \beta_{\lvert B\rvert}\}$	batches
$\Phi^\pi = \{\phi_1, \phi_2, \dots \phi_{max}\}$	asset properties dependent on process
$\mu = (H, S, P)$	cost severity levels: Hard, Soft, Penalty
$f(\phi, \pi, a_0, a_1, \mu)$	process cost for a process, property, pair of assets, and severity
K_H, K_S, K_P	coefficients of cost severity

3. RELATED WORK

To relate our work to state of the art we first situate our problem within job shop scheduling (JSS). Following that, we compare aspects of our solution to other approaches.

Problem Description Comparison

Branke et al [1] provide the most up to date comprehensive survey on JSS in the context of heuristic approaches. To the best of our knowledge, the similarity-based batch scheduling problem (SBBS) has not yet been addressed by the evolutionary computation community. We believe no benchmarks of the problem exist. It is a specific sub-class of JSS problem falling within the general class of job shop scheduling in batches. There are obvious parallels between processes and machines and between the multi-line and a series of machines. Our conjecture is that, like many multi-stage problems, it is generally NP-hard.

Branke et al [1] describe problems that pick jobs from an eligible list one by one. In subtle but important contrast, in similarity-based batch scheduling a batch of assets must be selected from a larger entry yard. This selection requirement is significant if a direct solution approach were to be taken because some sort of variable length representation would be required. For a dispatch rule supported by a priority function however, selection does not appear to impose extra requirements. That is, we believe the batch aspect of our problem is a unimportant specialization of dispatch rules that assign tasks to a machine one at a time.

Significant differences arise in terms of what task, job and machine information is relevant and available to the scheduler heuristic. Typically in JSS problems, (for example, see Table III in one survey [1] and Table 2 in another paper[6]), job properties such as processing time of an operation, ready time of a job, weight of a job are introduced. This reflects objectives centered around throughput or makespan. In contrast our problem introduces asset properties. This reflects objectives centered on batch "smoothness" or asset-to-asset differences. Additionally, our problem has no process attributes equivalent to machine attributes such as setup time. It also, notably, lacks any explicit objectives related to a global metric such as end to end throughput. There is no concept of end to end similarity or global configuration efficiency. A suite of local objectives (at the component level) must be optimized and, indirectly, this causes their collective

scheduling to be optimized because components are linked through exit and entry yards.

Solution Design Discussion

A majority of state of art solutions to JSS, per [1], generate an effective scheduler heuristic by simply applying candidate heuristics to a set of problem instances (the training instances), measuring their performance, and using this feedback to guide the search towards increasingly better heuristics. Genetic programming to evolve priority functions is one such example. Ant colony optimization is another [3, 4] In contrast to using a priority function, we use clustering, see Section 4.1, thus exploring the similarity aspect of our problem. The computational cost of optimizing weights used by clustering by using CMAES is arguably similar to the cost of using genetic programming to learn a priority function. CMAES likely requires less memory than genetic programming which uses a tree-based population. The computational costs of batching via clustering versus using a priority function are equivalent.

4. INTEGRATED SYSTEM

Overview The architecture of the integrated system, see Figure 3, has three components: a scheduler, a schedule simulator, and a hyper-optimization module.

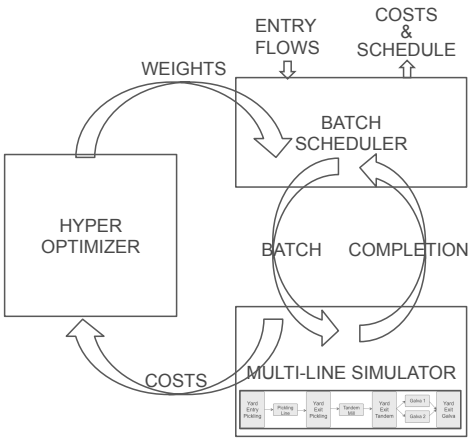

Figure 3: Integrated System Architecture

For each process, in two steps, the scheduler (1) selects a batch of assets from the entry yard and (2) sequences them. Detailed discussion of how it does this is deferred to Section 4.3. The scheduler adjusts the size of the batch to the number of assets that fit within an approximately specified weight capacity. In our problem instance a batch requires roughly 6 hours of processing. It passes each batch to the simulator and "waits for it to finish" during which new assets can enter the yard. When the simulator signals that the batch has been completed, it then starts selection and sequencing the next batch.

4.1 Simulator

The simulator, see Figure 4 models the specific multi-line and how it processes batches at each component. It updates down stream yard inventories that change due to processing. It is "triggered" each time the scheduler sends it a batch. Upon triggering, it simulates the timed movement of the batch through the process. For each asset-to-asset pair, it calculates whatever costs are incurred, tallying them, per property, on a batch basis. It advances simulated time to whenever the next batch will complete processing and informs the scheduler of that completion and its costs.

4.2 Hyper-Optimizer

The hyper-optimizer tunes the parameters of the scheduler with an evolutionary algorithm. This tuning is done online, when the scheduler is tasked to prepare schedules for some number of batches. For example, the scheduler might be optimized to provide the best schedules for a day (four approximately 6h batches) or for a single batch. We cloud scale the hyper-optimization algorithm, CMAES [5] to execute 30 runs in parallel so that the time for hyper-optimization is bounded by a single run. The time cost of a single run is dominated by the number of fitness evaluations of CMAES. Given our previous example of scheduling 4 batches daily, one fitness evaluation is a complete simulation run of four batches per process.

4.3 Scheduler

The scheduler needs to put similar assets beside each other but, as we have earlier explained, similarity is a complex concept. To address this the scheduler proceeds in two steps: 1. select 2. sequence.

Step 1: *CL* Selection The scheduler selects a batch from the entry yard prior to sequencing it by clustering the assets based on similarity.

1. The scheduler has a weight vector ω that has an entry for each property (Φ_i). It normalizes the property values for every asset in the yard and multiplies each property value by the appropriate weight in ω. It prepares a matrix Λ with the results where each column is an asset's weighted properties. Λ has dimensions ($|\Phi| \times |A|$) and ω has dimensions ($|\Phi| \times 1$). Note that weight vector ω is a hyper-parameter, i.e. optimized for the scheduler by CMAES.

2. The scheduler calls a machine learning algorithm named hierarchical clustering, HC[1] [2, 7] with Λ as an input argument. HC returns a distance matrix Δ and a dendrogram data structure T which represents the hierarchical clustering of the assets given their weighted properties, i.e., Λ. Δ, of dimensions $|A| \times |A|$, holds the distance computed by HC for each pair of assets given Λ.

3. The scheduler derives from T a batch β with procedure (a) using its clustering information. β is chosen so the quantity of assets meets the desired batch weight. To ensure smoothness between batches β includes the last asset a_{-1} of the previous batch (though this asset is not sequenced). The selection subroutine returns β as the selected (as yet un-sequenced) batch and Δ.

   (a) Find the previous asset a_{-1} and the cluster (subtree τ_{-1}) it belongs to (at the start pick the assets with lowest pairwise distance). Find the asset a_0 with lowest distance to the a_{-1} and add it to β and repeat. If all assets (leafs) in τ_0 have been added we traverse to the parent of τ_0^p. Stop when $\sum \beta_i < C$ or all assets have been added.

[1]http://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html#scipy.cluster.hierarchy.linkage

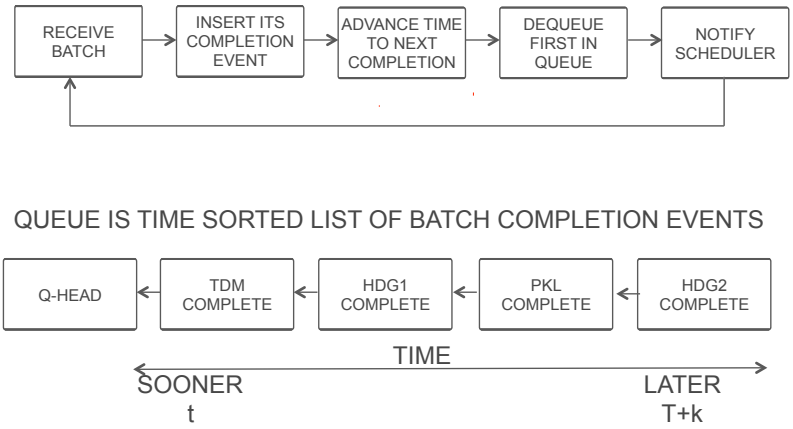

Figure 4: Block depiction of an example event-based simulator with four process: TDM, HDG1, PKL and HDG2. In this paper we use two processes: PKL and TDM.

Step 2: Sequencing

Option1: RANK Given β from Step 1, the scheduler uses the weighted distance matrix Δ to sequence the assets according to how far they are from the last asset of the previous batch (smallest distance to longest).

Option2: NN The first asset in the sequence is the one smallest in distance to the last asset of the previous batch, i.e., its nearest neighbor. The second asset is the one smallest in distance to the first, etc.

Discussion The weight vector ω is obviously critical to the scheduler's efficiency. If, say, its weights cause thickness similarity to be prioritized over width but the cost of a width mismatch is higher, the weights are poorly chosen. Conversely, well-determined weights accurately reflect the indirect relationship between costs and properties bearing the overall goal of cost-efficient multi-line sequencing in mind. Rather than analytically determine the weights they are tuned via the hyper-optimization algorithm, see Figure 3. The hyper-optimizer algorithm optimizes the weights to minimize the costs of multiple cost-efficiency objectives. We provide parameter settings for CMAES in Section 5.2.

5. EXPERIMENTAL EVALUATION

5.1 Problem Instance

We demonstrate our approach with the single problem instance made available to us. The simplified multi-line has 2 components, named PKL and TDM respectively. The exit yard of the PKL component joins with the entry yard of the TDM component. For the PKL process there are 3 relevant similarity properties: thickness, width, and length. For the TDM process there are 3 relevant similarity properties: thickness, width and strength. The severity coefficients are not precisely calibrated to reflect how the actual process constraint violation costs would interact with property distributions of the assets. This implies that while it is possible to determine if one option is superior among all of them, in the future, with calibration, rankings are likely to change. Tables 2, 3 and 4 provide explicit details. We minimize a single objective by summing all weighted costs, integrating across severity, property, and all processes. Our cost functions are

Table 2: Problem Instance Descriptions

Processes	Π	(PKL, TDM) exit(PKL) = entry(TDM)
Properties PKL TDM	 Φ^{PKL} Φ^{TDM}	 (length l, width w, thickness d) (width w, thickness d, strength s)
Coefficients	Severity	$K_H = 1000, K_S = 100, K_P = 1$

Table 3: PKL Process Cost Functions. Cost is multiplied by severity.

ID	ϕ_i	Severity	Cost logic (ϕ_0,ϕ_1)
$f1_{PKL}$	l	K_S	1500 if $(l_{-3} + l_{-2} + l_{-1} + l_0) < 2000m$
$f2_{PKL}$	w	K_H	1 if $\lvert w_0 - w_1 \rvert \leq 200$
$f3_{PKL}$	w	K_S	1 if $170 < \lvert w_0 - w_1 \rvert < 200$
$f4_{PKL}$	d	K_H	1 if $\lvert d_0 - d_1 \rvert / d_0 > 0.5$
$f5_{PKL}$	d	K_P	1 if $k_d \lvert d_0 - d_1 \rvert$

$\mathbf{f_{PKL}} = \{f1_{PKL}, f2_{PKL}, f3_{PKL}, f4_{PKL}, f5_{PKL}\}, \mathbf{f_{TDM}} = \{f1_{TDM}, f2_{TDM}, f3_{TDM}, f4_{TDM}, f5_{TDM}, f6_{TDM}\}$ and they are calculated as in Eq. (9).

5.2 Evaluation Procedure

We use 7 days of the entry yard data provided to us. We select the weight tuning (i.e. hyper-optimization) interval to be daily and we schedule 4 batches in roughly a day. With each daily set of optimized weights we compute the batch costs averaged up to that day since each prior day determines the inventories for the next. We perform 30 runs of CMAES using a Python package[2] with 100 fitness evaluations per run to obtain 30 sets of optimized weights daily to obtain more robust statistics on average batch cost. The population size, chosen by the Python package is 9 and the weight vector ω has 6 property dimensions (3 for each of PKL and TDM). We do not change any default parameters settings used by the package.

[2]https://www.lri.fr/~hansen/cmaes_inmatlab.html

Table 4: TDM Process Cost Functions. Cost is multiplied by severity.

ID	ϕ_i	Severity	Cost logic (ϕ_0,ϕ_1)
$f1_{TDM}$	w	K_H	1 if $\lvert w_0 - w_1\rvert \leq 5$ or $\lvert w_1 - w_0\rvert \leq 150$
$f2_{TDM}$	w	K_S	1 if $\lvert w_0 - w_1\rvert < 2$
$f3_{TDM}$	w	K_P	1 if $\lvert w_0 - w_1\rvert$ when $w_1 > w_0$
$f4_{TDM}$	d	K_H	1 if $\lvert d_0 - d_1\rvert / d_0 > 0.25$
$f5_{TDM}$	d	K_P	1 if $k_d \lvert d_0 - d_1\rvert$, $k_d = 1$
$f6_{TDM}$	s	K_P	1 if $k_d \lvert s_0 - s_1\rvert$, $k_d = 10$

For hierarchical clustering we use the algorithm in package SCIPY[3]. We use weighted Euclidean[4] distance to measure similarity between assets. Distance between clusters is based on the nearest point algorithm.

5.3 Comparisons

We define two **baseline selection heuristics**.

1. TOP sorts the yard assets on a single property and selects assets from the start of the sort downwards.

2. RDM randomly selects assets from the yard.

We define a **baseline sequencing heuristic** SORT that sorts the assets on a single property.

Table 6 shows the results of our experiments. We performed a Wilcoxon rank sum test[5] and the difference between experiments were all statistically significant for a significance level of $\alpha = 0.05$.

With our problem instance we first run the TOP selection and SORT sequencing (TOP+SORT) on 9 combinations of single properties, one each for PKL and TDM. We obtain the best result of 12.169 when width is the sort property for both PKL and TDM. This implies, for this data and multi-line configuration, width for both processes is the most influential property affecting cost. This knowledge provides some insight into the specific property values of the assets we are trying to schedule and how they interact with specific cost parameters and severity weights. This knowledge cannot be derived from the costs alone. As additional information, we note that the difference in average batch cost between TOP+SORT on the worst set of properties (length for PKL combined with strength for TDM) is 12.921, which is only 6% different from the best. This reveals the multi-line definition and sample assets we are investigating to be quite limited in scope for optimization. This could potentially change if severity coefficients were calibrated.

We next run the RDM selection and SORT sequencing (RDM+SORT) using the width property for both the PKL and TDM processes. We find a increase in average batch cost to 12.216 ± 0.211. Nether RDM+SORT and TOP+SORT exploit hyper-optimization (because they do not rely upon clustering or information from the distance score matrix. Not significantly different, they represent the best performance that can be obtained without hyper-optimization.

[3]http://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html#scipy.cluster.hierarchy.linkage

[4]weighted Minkowski with p–norm 2

[5]http://docs.scipy.org/doc/scipy-0.16.1/reference/generated/scipy.stats.ranksums.html

Table 5: Combined selection and sequencing heuristics. The last two heuristics use no baseline heuristics while the ones above them use one or more.

NAME	SELECTION	SEQUENCING
RDM+SORT	RDM	SORT
TOP+SORT	TOP	SORT
TOP+RANK	TOP	RANK
TOP+NN	TOP	NN
CL+SORT	CL	SORT
CL+RANK	CL	RANK
CL+NN	CL	NN

Table 6: Results for each selection and sequencing heuristic. Cost is calculated as the average for each 6h batch.

Setup	Average 6h Cost
TOP+SORT	12.169
RDM+SORT	12.216 ±0.211
TOP+RANK	9.580 ± 0.452
TOP+NN	8.491 ± 0.168
CL+RANK	10.032 ± 0.501
CL+NN	8.789 ±0.313
CL+SORT	10.006 ± 0.063

To isolate sequencing from selection effects, we next pair the TOP selection heuristic with both RANK and NN. Whether RANK or NN will outperform SORT will depend, in general on the multi-line definition and sample assets being scheduled. We find that TOP+NN has on average the significantly better performance: 8.491 ± 0.168 vs 9.580 ± 0.452 for TOP+RANK.

We next assess our hierarchical clustering (abbreviated CL) as a selection heuristic paired with either SORT, NN or RANK. When selecting batches with clustering and then sequencing the selected assets, the CL+NN has the best average performance, 8.789 ± 0.313. We note however that TOP+NN has the best performance (over CL+NN p-value 0.00013). For the CL+SORT and CL+RANK the performance appears almost equal but the difference is significant. Thus, it seems that clustering can help reduce the cost. although, it is very important to use the correct sequence heuristic after selection.

These results indicate that optimizing weights reduces the cost. They also shows that using the nearest neighbor to select the next coil can help performance. Given these rankings could shift when different multi-lines are defined or different asset flows are at hand, if cloud computing were available and inexpensive, using the cloud to try every heuristic in combination before scheduling would be ideal.

One question that arose is whether weights optimized for one day would be generally robust and therefore effective on later days. If so, the interval between hyper-optimizing could be lengthened. Proceeding pessimistically, we sequestered the first day weights from the poorest performing clustering-based heuristic which used RANK sequencing. We then used them to schedule the following 6 days with no hyper-optimization. The performance for weights taken from CL+RANK then run over the entire week was 11.045. These very poor results would seem to indicate that the weights are very tightly optimized at a daily interval. This negative result, in turn, motivated us to shorten the hyper-optimization interval to its minimum - at every batch. Given concern that

the hyper-optimization must be fast enough to not overrun the time it takes to process a batch, we again proceeded pessimistically by selecting the algorithm that runs for the longest time. This was CL+NN. When run at sub-batch speed, the hyper-optimization generated weights that drove costs down even lower than at the daily level. We obtained an average batch cost of 8.348 ± 0.204.

Finally, we examined the weights themselves. Our expectation was that we would see high weights on the width property of each process because we had seen these properties influence batching when we selected based on sorting on a single property. We did see this for solutions that used TOP where the selection was based on one property. However for experiments that used clustering for selection we saw inconsistent weights. We will continue to work on correctly interpreting $\omega's$. Our intuition is that, because of clustering, the weights are a more complex function of data as well as of the constants in the problem costs.

6. SUMMARY

We have presented a job shop scheduling problem previously undocumented in evolutionary computation literature. Its objective is to minimize configuration costs that depend on the sliding pairwise similarity between two assets ordered one after the other in a processing batch. We therefore needed to learn weights that integratively express complex asset similarity. Our proposed solution is a three component scheduling system: simulator, scheduler and hyper-optimizer where the scheduler first selects a group of assets then sequences them. The scheduler relies upon hierarchical clustering to select, from an entry yard, assets for a batch that are similar to each other in a weighted multi-dimensional sense. It then references weighted distance information to sequence the assets. We propose two heuristics that either use a ranking based on distance to the last asset of the previous batch or that generate a pairwise nearest neighbor ordering. The weights used by the scheduler are optimized online with an evolutionary algorithm. Given straight forward cloud-based parallelization, the time cost of scheduling a batch while optimizing and simulating is shorter than the time it takes the batch to travel through its process, allowing online optimization that supports efficient scheduling.

7. FUTURE WORK

Limitations of pairwise similarity-based scheduling like in this contribution arise if there is a significant cost that involves two or more assets. For example, cost function $cf1$ of Table 3, describes a penalty if four assets' total length is less than a threshold. As well, it is reasonable (and typical) to have some time constraint or objective in a scheduling problem. How to integrate similarity-based costs with time based ones remains an open question we would like to tackle soon. The computing time will be investigated for how the solution quality evolves along through the computing time to assess the stopping criteria. Furthermore, the algorithms should be tested on more instances to investigate how the performance changes. In addition, more constraint-handling techniques in evolutionary algorithms will be investigated. Finally, we will investigate how the algorithms perform on other data and multi-line instances.

8. REFERENCES

[1] J. Branke, S. Nguyen, C. Pickardt, and M. Zhang. Automated design of production scheduling heuristics: A review. *IEEE Trans. Evolutionary Computation*, 20(1):110 – 124, 2015.

[2] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In J. Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 434–444. ACM, 1988.

[3] S. Fernandez, S. Alvarez, D. Díaz, M. Iglesias, and B. Ena. Scheduling a galvanizing line by ant colony optimization. In M. Dorigo, M. Birattari, S. Garnier, H. Hamann, M. A. M. de Oca, C. Solnon, and T. Stützle, editors, *Swarm Intelligence - 9th International Conference, ANTS 2014, Brussels, Belgium, September 10-12, 2014. Proceedings*, volume 8667 of *Lecture Notes in Computer Science*, pages 146–157. Springer, 2014.

[4] S. Fernandez, S. Alvarez, E. Malatsetxebarria, P. Valledor, and D. Díaz. Performance comparison of ant colony algorithms for the scheduling of steel production lines. In S. Silva and A. I. Esparcia-Alcázar, editors, *Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015, Companion Material Proceedings*, pages 1387–1388. ACM, 2015.

[5] N. Hansen. The CMA evolution strategy: a comparing review. In J. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, editors, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, pages 75–102. Springer, 2006.

[6] R. Hunt, M. Johnston, and M. Zhang. Evolving "less-myopic" scheduling rules for dynamic job shop scheduling with genetic programming. In *Genetic and Evolutionary Computation Conference, GECCO '14, Vancouver, BC, Canada, July 12-16, 2014*, pages 927–934, 2014.

[7] M. J. Kearns, Y. Mansour, and A. Y. Ng. An information-theoretic analysis of hard and soft assignment methods for clustering. In D. Geiger and P. P. Shenoy, editors, *UAI '97: Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence, Brown University, Providence, Rhode Island, USA, August 1-3, 1997*, pages 282–293. Morgan Kaufmann, 1997.